Examen DeliverUS - Modelo B - Marzo

Recuerde que DeliverUS está descrito en: https://github.com/IISSI2-IS-2025

Enunciado del examen

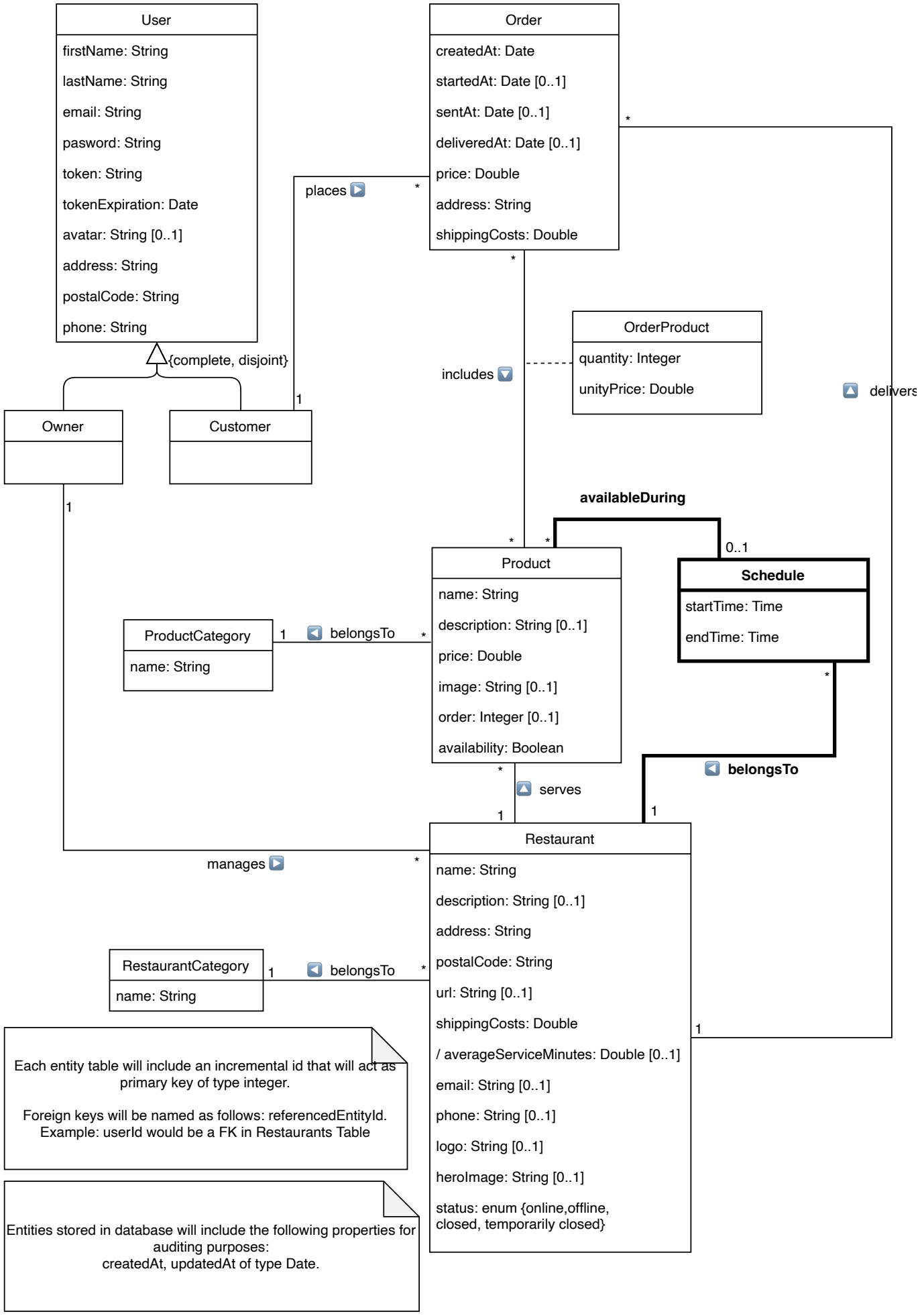
Tenemos que incorporar un nuevo requisito en la aplicación DeliverUS que consiste en que los productos estarán disponibles en unos ciertos horarios.

¿Qué son los horarios (Schedules)?

En DeliverUS, los horarios (Schedules) representan intervalos concretos de tiempo (con hora de inicio y hora de fin en formato HH:mm:ss) en los que un restaurante o ciertos productos del restaurante están disponibles para pedidos. Esto permite definir claramente cuándo se pueden realizar pedidos de ciertos productos.

Los horarios son útiles, por ejemplo, para ofrecer menús específicos de desayuno, almuerzo o cena, y para establecer horas concretas en las que ciertos productos pueden estar disponibles.

Cada propietario de restaurante podrá definir los horarios para cada uno de sus restaurantes, y por lo tanto, cada producto se podrá asociar a uno de los horarios registrados para el restaurante al que pertenece. Se nos ha proporcionado el siguiente modelado conceptual:



Es necesaria la implementación de los siguientes requisitos funcionales:

RF1. Listado de horarios de restaurante

Como usuario, quiero listar los horarios de un restaurante para poder gestionarlos fácilmente

Ruta: `GET /restaurants/:restaurantId/schedules`

Pruebas de aceptación:

- Devuelve un array con todos los horarios del restaurante especificado. El objeto json que devuelve debe tener la forma del ejemplo siguiente:

```
[{
    id: 1,
    startTime: "08:00:00",
    endTime: "11:00:00",
    restaurantId: 1,
    createdAt: "2025-03-12T16:33:20.000Z",
    updatedAt: "2025-03-12T16:33:20.000Z",
},
{
    id: 2,
    startTime: "12:00:00",
    endTime: "15:00:00",
    restaurantId: 1,
    createdAt: "2025-03-12T16:33:20.000Z",
    updatedAt: "2025-03-12T16:33:20.000Z",
}]
```

- Devuelve código `404` si el restaurante no existe.

RF2. Creación de horario

Como propietario, quiero crear un nuevo horario para gestionar la disponibilidad de mis productos

Ruta: `POST /restaurants/:restaurantId/schedules`

Pruebas de aceptación:

- El horario creado contiene los atributos `startTime` y `endTime` correctamente definidos. Recibe un objeto json en el cuerpo de la petición con la forma del siguiente ejemplo:

```
    {
        startTime: '08:00:00',
        stopTime: '12:00:00'
    }
```

- Devuelve código `401` si el usuario no está autenticado.
- Devuelve código `403` si el usuario no es propietario del restaurante.

- Devuelve código `404` si el restaurante no existe.
- Devuelve código `422` si faltan atributos o si las validaciones fallan:
    - `startTime` y `endTime` tienen formato de hora correcto (se le proporcionan funciones auxiliares para ello).
    - `endTime` debe ser mayor que `startTime`.

RF3. Edición de horario

Como propietario, quiero editar un horario existente para ajustar la disponibilidad de mis productos

Ruta: `PUT /restaurants/:restaurantId/schedules/:scheduleId`

Pruebas de aceptación:

- Actualiza correctamente el horario especificado. Recibe un objeto json en el cuerpo de la petición con la forma del siguiente ejemplo:

```
{
    startTime: '08:00:00',
    stopTime: '12:00:00'
}
```

- Devuelve código `401` si el usuario no está autenticado.
- Devuelve código `403` si el usuario no es propietario del restaurante.
- Devuelve código `404` si el restaurante o el horario no existen.
- Devuelve código `422` si faltan atributos o si las validaciones fallan:
    - `startTime` y `endTime` tienen formato de hora correcto (se le proporcionan funciones auxiliares para ello).
    - `endTime` debe ser mayor que `startTime`.

RF4. Borrado de horario

Como propietario, quiero eliminar un horario que ya no necesito para ajustar los horarios de mi restaurante

Ruta: `DELETE /restaurants/:restaurantId/schedules/:scheduleId`

Pruebas de aceptación:

- Borra correctamente el horario especificado.
- Devuelve código `401` si el usuario no está autenticado.
- Devuelve código `403` si el usuario no es propietario del restaurante.
- Devuelve código `404` si el restaurante o el horario no existen.

RF5. Productos Activos y detalles de restaurante

Como usuario, quiero ver todos los detalles de un restaurante incluyendo solo los productos activos según el horario actual para consultar los productos disponibles

Ruta: `GET /restaurants/:restaurantId/showWithActiveProducts`

Pruebas de aceptación:

- Devuelve el restaurante incluyendo solo los productos cuyo horario incluye la hora actual del servidor. Devuelve un objeto json en el cuerpo de la petición con la forma del siguiente ejemplo:

```
{
id: 25,
name: "Bodeguita Antonio Romero",
description: "Comida Sevillana",
//resto de propiedades del restaurante

products:  // array de productos activos en el momento de la petición
    [
        {
        id: 30,
        name: "Producto activo",
        description: "Debe mostrarse",
        price: 10,
        //resto de propiedades del producto
        },
        {
        id: 33,
        name: "Otro Producto activo",
        description: "Debe mostrarse",
        price: 5,
        //resto de propiedades del producto
        }
    ]
```

- Productos con `scheduleId` nulo no se consideran activos.
- Devuelve código `404` si el restaurante no existe.

RF6. Selección de horario en creación de producto

Como propietario, quiero poder especificar un horario para el producto para ajustar la disponibilidad de mis productos

Ruta: `POST /products`

Pruebas de aceptación:

- El horario es opcional. Recibe un objeto json en el cuerpo de la petición con la información del producto y opcionalmente la propiedad `scheduleId` con un valor
- Si se especifica un horario, el horario debe pertenecer al restaurante del producto
- Resto de comprobaciones ya existentes

RF7. Selección de horario en edición de producto

Como propietario, quiero poder especificar un horario para el producto para ajustar la disponibilidad de mis productos

Ruta: `PUT /products/:productId`

Pruebas de aceptación:

- El horario es opcional. Recibe un objeto json en el cuerpo de la petición con la información del producto y opcionalmente la propiedad `scheduleId` con un valor
- Si se especifica un horario, el horario debe pertenecer al restaurante del producto
- Resto de comprobaciones ya existentes

Ejercicios

1. Migraciones necesarias (1 punto)

Cree y modifique las migraciones necesarias para implementar el modelado conceptual anterior.

Se le proporciona el fichero `/src/database/migrations/20210718065004-create-schedules.js` para la migración de `Schedule`.

---

2. Modelos y cambios necesarios en modelos existentes (1 punto)

Cree y modifique los modelos necesarios para implementar el modelado conceptual anterior. Para el modelo de `Schedule` se le proporciona el fichero `/src/models/Schedule.js` para el modelo de `Schedule`.

---

3. Rutas de Schedule (1 punto)

Implemente las siguientes rutas:

- RF1: **GET** `/restaurants/:restaurantId/schedules`
- RF2: **POST** `/restaurants/:restaurantId/schedules`
- RF3: **PUT** `/restaurants/:restaurantId/schedules/:scheduleId`
- RF4: **DELETE** `/restaurants/:restaurantId/schedules/:scheduleId`

Se le proporciona el fichero `/src/routes/ScheduleRoutes.js` para definir las rutas de `Schedule` y recuerde incorporar los middlewares necesarios en cada ruta.

---

4. Validación para Schedule (1 punto)

Implemente las reglas de validación para creación y actualización de `Schedule` teniendo en cuenta lo especificado en el modelo conceptual y que el `endTime`es posterior a `startTime`. Se espera que ambas propiedades estén definidas en edición.

Se le proporciona el fichero `/src/controllers/validation/ScheduleValidation.js` para definir las validaciones de `Schedule`. Se le proporicionan además dos funciones de apoyo:

- `validateTimeFormat(value)` (comprueba formato HH:mm:ss)
- `validateEndTimeAfterStartTime(endTime, { req })` (comprueba que `endTime` es posterior a `startTime`)

---

5. Controlador de Schedule (2 puntos)

Implemente las funciones necesarias para cumplir con RF1, RF2, RF3, RF4.

Se le proporciona el fichero `/src/controllers/ScheduleControler.js`.

---

6. Validación de creación de Producto (1 punto)

Valide la creación de producto teniendo en cuenta la relación definida en el modelo conceptual y el RF6.

Se le proporciona el fichero `/src/controllers/validation/ProductValidation.js` con el prototipo de la función `checkScheduleBelongsToRestaurantOnCreate`.

---

7. Validación de edición de Producto (1,5 puntos)

Valide la edición de producto teniendo en cuenta la relación definida en el modelo conceptual y el RF7.

Se le proporciona el fichero `/src/controllers/validation/ProductValidation.js` con el prototipo de la función `checkScheduleBelongsToRestaurantOnUpdate`.

---

8. Detalle de restaurante incluyendo sólo productos activos (1,5 puntos)

Implemente la siguiente ruta:

- RF5: **GET** `/restaurants/:restaurantId/showWithActiveProducts`

Implemente asimismo la función `showWithActiveProducts` cuyo prototipo se le proporciona en el fichero `/src/controllers/RestaurantController.js` para cumplir el RF5.

- Productos con `scheduleId` nulo NO se consideran activos automáticamente.
- La consulta debe considerar la hora actual del servidor.

Puede usar `new Date().toTimeString().split(' ')[0]` para obtener el tiempo actual en formato `HH:mm:ss`.

Información adicional importante

- **Las rutas y validaciones deben respetarse exactamente como aquí se describen, ya que los tests automáticos se basan en estas especificaciones.**
- **No modificar los tests.** El fichero de test `/tests/e2e/schedules.test.js` se comprueban explícitamente las rutas, estructuras de datos, validaciones y asociaciones descritas anteriormente.

Procedimiento de entrega

1. Borrar las carpetas **node_modules** de backend.
2. Crear un ZIP que incluya todo el proyecto. **Importante: Comprueba que el ZIP no es el mismo que te has descargado e incluye tu solución**
3. Avisa al profesor antes de entregar.
4. Cuando el profesor te dé el visto bueno, puedes subir el ZIP a la plataforma de Enseñanza Virtual. **Es muy importante esperar a que la plataforma te muestre un enlace al ZIP antes de pulsar el botón de enviar**. Se recomienda descargar ese ZIP para comprobar lo que se ha subido. Un vez realizada la comprobación, puedes enviar el examen.

Preparación del entorno

a) Windows

- Abra un terminal y ejecute el comando `npm run install:all:win`.

b) Linux/MacOS

- Abra un terminal y ejecute el comando `npm run install:all:bash`.

Ejecución

Backend

- Para **rehacer las migraciones y seeders**, abra un terminal y ejecute el comando

```
npm run migrate:backend
```

- Para **ejecutarlo**, abra un terminal y ejecute el comando

```
npm run start:backend
```

Depuración

- Para **depurar el backend**, asegúrese de que **NO** existe una instancia en ejecución, pulse en el botón `Run and Debug` de la barra lateral, seleccione `Debug Backend` en la lista desplegable, y pulse el botón de *Play*.

Test

- Para comprobar el correcto funcionamiento de backend puede ejecutar el conjunto de tests incluido a tal efecto. Para ello ejecute el siguiente comando:

```
npm run test:backend
```

Advertencia: Los tests no pueden ser modificados.

Problemas con los puertos

En ocasiones, los procesos de backend, con o sin depuración, puede quedarse bloqueado sin liberar los puertos utilizados, impidiendo que puedan ejecutarse otros procesos. Se recomienda cerrar y volver a iniciar VSC para cerrar dichos procesos.